LAB2 : DEEP LEARNING FOR NLP

Neural Network : Classification with PyTorch

Florian Bertelli, Sébastien Warichet

ABSTRACT

In the following document our aim is to implement a neural network, using pytorch. The task will be to predict if one reviews is positive or negative, using a different architectures of deep neural networks (with hidden layers).

Data description

- The data are downloaded from the IMDB database, we will used the 5 000 first sentences, each sentence being associated with the corresponding label (positive/negative review).

Summary

- This lab contains a full implementation of a convolution neural network (in the class "Conv classifier"), with the training of an embedding and some linear classifiers using :
  * x is an input sentence that is represented as an vector of word id, each word id represent a different word of corpus
  * *vocabsize* and *embeddingdim* are the parameters of the embedding
  * *pooltype* is the parameter that define which pool (mean or max) is applied after convolution
  * *listsize* is a list parameters that define the number of convolution (length of list size) and the size of each convolution. * *featsize* is a parameter that define the size of the feature space. * sigmoid transforms the output of the last Linear into a value between 0 and 1
  * *output* is a value that contains the probability of positive or negative review. If $p > 0.5$, the prediction is positive and if $p < 0.5$ the prediction is negative.
- Our loss function, that we will try to minimize during the training is : BCELoss

$$l(x, y) = -(y * log(x) + (1 - y) * log(1 - x))$$

We've used this function and not BCELoss with Logits because BCELoss With logits includes directly the sigmoid activation, and we've used sigmoid activation in the last layer.

- Then it contains a different architectures of neural networks and with the activation function *sigmoid*. This will enables us to see the effect of using different architectures for this really simple task.
- We've used an adaptive learning rate in order to reach the global minimum, by dividing it by a certain factor every 5 epochs.
- Lastly, we shuffle the data before each training, and keep the best model on the dev set over all epoch for the testing.

Test and results

In this first part we used a very simple architecture

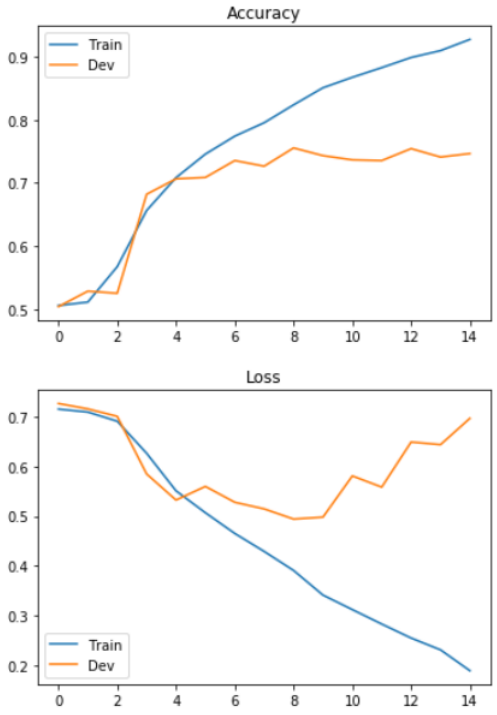
- One embedding Layer (numEmbeddings = vocabSize, embeddingDim= 5)
- One hidden layer Layer (hiddenSize*embeddingDim, featsize= 5)
- One pooling layer(max or mean)
- Output Layer Sigmoid

Furthermore, in the second part we're dealing with a embedding already trained, issue from the glove data set in 50dimensions.

We obtained the following graphs : the first one is the accuracy, for and the second one the loss against the number of epochs.

Filter of size 3, feature space of size 5, max pooling

The following graph shows you the accuracy and loss for the training set and the dev set against the number of epoch for the model mentioned above.

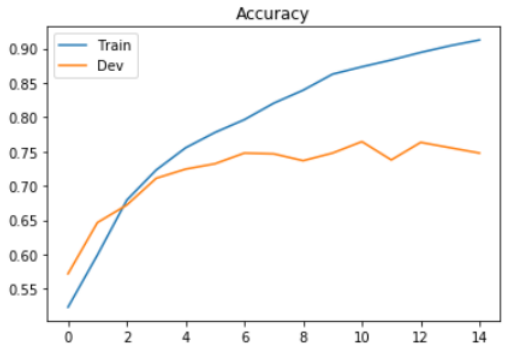


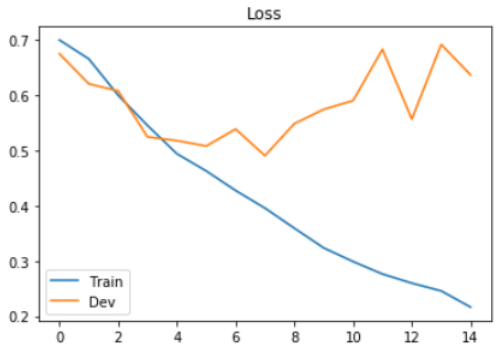
We can see that the model is overfitting from epoch 4th to the end. Indeed, our model doesn't manage to generalize well on dev data.

However we get an accuracy on the test set of 0.745 which is great but not really better than last time.

Filter of size 3, feature space of size 5, mean pooling

The following graph shows you the accuracy and loss for the training set and the dev set against the number of epoch for the model mentioned above.





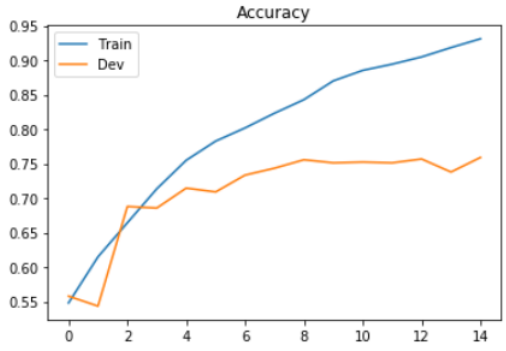
The difference between this model and the last one is that this ones takes into account every pixel to compute the mean during pooling, and the max only keeps the higher one.

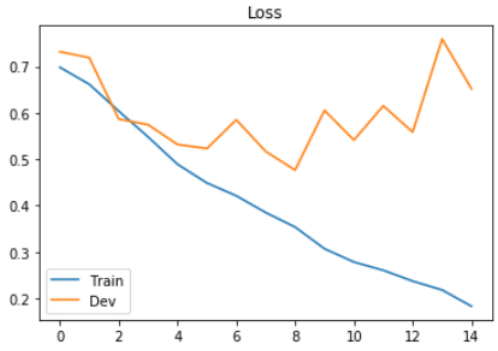
We can see that the model is overfitting from epoch 4th to the end. Indeed, our model doesn't manage to generalize well on dev data.

However we get an accuracy on the test set of 0.74 which is great, but the almost the same than the max pooling.

Filter of size 2, feature space of size 5, max pooling

The following graph shows you the accuracy and loss for the training set and the dev set against the number of epoch for the model mentioned above.



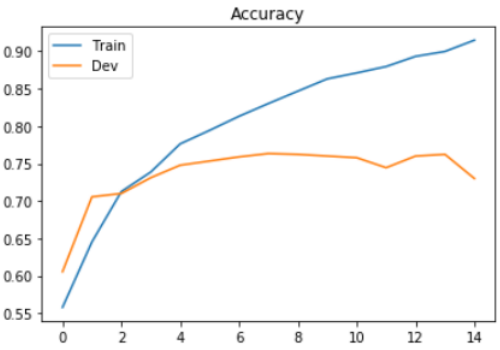


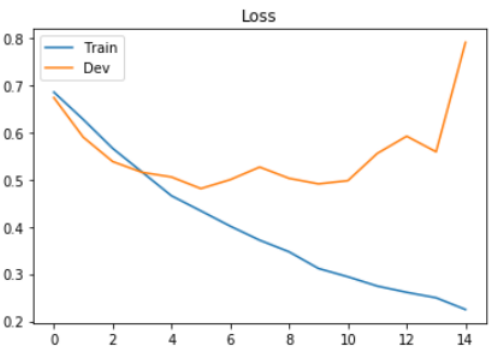
We can see that the model is overfitting from epoch 4th to the end. Indeed, our model doesn't manage to generalize well on dev data.

However we get an accuracy on the test set of 0.757 which is great with 1% more than the precedent models. This is a great start.

Filter of size 2, feature space of size 5, mean pooling

The following graph shows you the accuracy and loss for the training set and the dev set against the number of epoch for the model mentioned above.





We can see that the model is overfitting from epoch 4th to the end. Indeed, our model doesn't manage to generalize well on dev data.

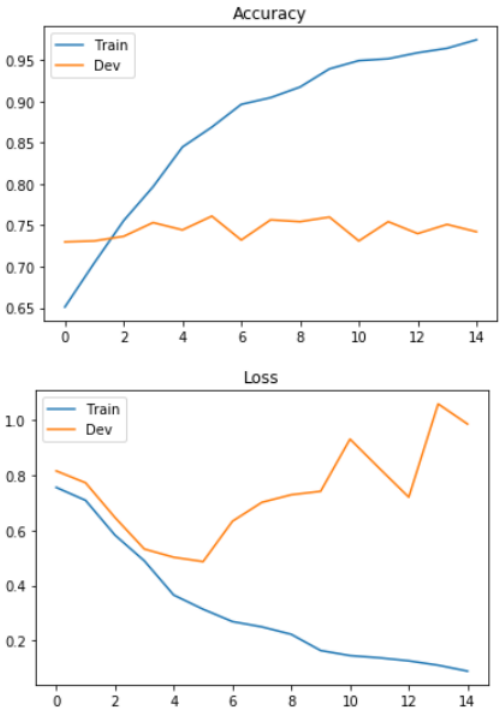
However we get an accuracy on the test set of 0.756 which is great with 1% more than the precedent models. This is a great start.

Analyses of these results

So we can see that the according to the results above, the mean or max pooling function have not a strong impact on this classification problem. Indeed, the results are quite the same independently of the pooling function.
Now if we get interested in the size of the kernel we can clearly see that this has an impact on the results. Indeed, a kernel of size 2 gets better results than a kernel of size 3. This means that the relation between words are more important to be analysed in term of pairs than triplet.

Pre Trained embedding

In this art we're dealing with a embedding already trained, issue from the glove data set in 50dimensions, we hope it will gives us interesting results because it provides a better information than a self trained embedding network. There are the results



We can clearly see that now we overfit at the 2 step, that our model can learn very fast but it is quite limited on this classification task. However, we get an even better result than every model before : 0.778 ie almost 78% of accuracy on the test this is quite interesting. Thanks to this pre trained model we've got 2% more than before, which is very interesting.

Conclusion

To conclude, we saw that to do this basic text classification task, a CNN works well but not really better than the simple model we've used last time. However, our best performance on test set is better than before (78 percent maximum). We can wonder how to get better results. An idea can be to have a larger training set because of data sparsity in NLP. Indeed generally many words are present a few times, so we've chance to never meet them during training, but only in testing, and with our implementation all unseen words are seen as "UNK" so this doesn't help us at all to train, it can make things even worse.